DATA IN MOTION

SQL Case Study Tiny Shop Sales

-- Case Study Questions

--1) Which product has the highest price? Only return a single row.

SELECT product\_name AS product, price

FROM products

WHERE price = (SELECT MAX(price) AS highest\_price

FROM products)

Explanation:

The query selects the **product\_name** and **price** from the **products** table.

The WHERE clause filters the rows by selecting only the rows where the price is equal to the maximum price obtained from a subquery that calculates the maximum price.

Output:

|  |  |
| --- | --- |
| product | price |
| Product M | 70 |

--2) Which customer has made the most orders?

SELECT CONCAT(c.first\_name, ' ', c.last\_name) AS customer, oi.quantity AS order\_quantity

FROM customers c

JOIN orders o

ON c.customer\_id = o.customer\_id

JOIN order\_items oi

ON o.order\_id = oi.order\_id

WHERE quantity = (SELECT MAX(quantity)

FROM order\_items)

GROUP BY CONCAT(c.first\_name, ' ', c.last\_name), oi.quantity

Explanation:

The query selects the concatenated **first\_name** and **last\_name** columns as **customer** and the **quantity** column as **order\_quantity**.

It joins the **customers**, **orders**, and **order\_items** tables on the appropriate keys.

The WHERE clause filters the rows to select only the rows where the quantity is equal to the maximum quantity obtained from a subquery.

Output:

|  |  |
| --- | --- |
| customer | order\_quantity |
| John Doe | 4 |

--3) What’s the total revenue per product?

SELECT product\_name, SUM(price\*quantity) AS revenue

FROM products p

JOIN order\_items oi

ON oi.product\_id = p.product\_id

GROUP BY product\_name

ORDER BY revenue DESC

Explanation:

The query selects the **product\_name** and calculates the total revenue using the **SUM()** function on the price multiplied by the quantity.

It joins the **products** and **order\_items** tables on the product ID.

The results are grouped by **product\_name** and ordered by the revenue in descending order.

Output:

|  |  |
| --- | --- |
| product\_name | revenue |
| Product M | 420 |
| Product J | 330 |
| Product F | 210 |
| Product L | 195 |
| Product K | 180 |
| Product C | 160 |
| Product I | 150 |
| Product B | 135 |
| Product H | 135 |
| Product G | 120 |
| Product E | 90 |
| Product D | 75 |
| Product A | 50 |

--4) Find the day with the highest revenue.

WITH CTE\_rev AS(

SELECT SUM(price\*quantity) AS revenue, order\_date

FROM products p

JOIN order\_items oi

ON oi.product\_id = p.product\_id

JOIN orders o

ON o.order\_id = oi.order\_id

GROUP BY order\_date)

SELECT MAX(revenue) AS highest\_rev, order\_date

FROM CTE\_rev

GROUP BY order\_date

HAVING MAX(revenue) = (SELECT MAX(revenue)

FROM CTE\_rev)

Explanation:

The query uses a Common Table Expression (CTE) to calculate the total revenue per day by joining the necessary tables and grouping by the **order\_date**.

The main query then selects the maximum revenue and corresponding order date using the MAX() function.

The HAVING clause ensures that only the rows with the maximum revenue are returned.

Output:

|  |  |
| --- | --- |
| highest\_rev | order\_date |
| 340 | 2023-05-16 |

--5) Find the first order (by date) for each customer.

SELECT CONCAT(first\_name, ' ', last\_name) AS customer, order\_date

FROM

(

SELECT first\_name, last\_name, order\_date,

ROW\_NUMBER() OVER (PARTITION BY c.first\_name, c.last\_name ORDER BY order\_date) AS row\_num

FROM orders o

JOIN customers c

ON o.customer\_id = c.customer\_id

) AS t

WHERE row\_num = 1

ORDER BY order\_date

Explanation:

The query uses a subquery to assign a row number to each order by date for each customer using the **ROW\_NUMBER()** function.

The outer query selects only the rows with **row\_num = 1**, representing the first order for each customer, and orders them by the order date.

Output:

|  |  |
| --- | --- |
| customer | order\_date |
| John Doe | 2023-05-01 |
| Jane Smith | 2023-05-02 |
| Bob Johnson | 2023-05-03 |
| Alice Brown | 2023-05-07 |
| Charlie Davis | 2023-05-08 |
| Eva Fisher | 2023-05-09 |
| George Harris | 2023-05-10 |
| Ivy Jones | 2023-05-11 |
| Kevin Miller | 2023-05-12 |
| Lily Nelson | 2023-05-13 |
| Oliver Patterson | 2023-05-14 |
| Quinn Roberts | 2023-05-15 |
| Sophia Thomas | 2023-05-16 |

--6) Find the top 3 customers who have ordered the most distinct products

SELECT TOP(3) CONCAT(first\_name, ' ', last\_name) AS customer, quantity

FROM

(

SELECT c.first\_name, c.last\_name, COUNT(DISTINCT oi.product\_id) AS quantity,

ROW\_NUMBER() OVER (ORDER BY COUNT(DISTINCT oi.product\_id) DESC) AS row\_num

FROM order\_items oi

JOIN orders o

ON o.order\_id = oi.order\_id

JOIN customers c

ON o.customer\_id = c.customer\_id

GROUP BY c.first\_name, c.last\_name

) AS t

WHERE row\_num <= 3

ORDER BY quantity DESC;

Explanation:

The query uses a subquery to calculate the count of distinct products ordered by each customer and assigns a row number to each count using the **ROW\_NUMBER()** function.

The outer query selects the top 3 rows based on the row number and orders them by the quantity in descending order.

Output:

|  |  |
| --- | --- |
| customer | quantity |
| Jane Smith | 3 |
| Bob Johnson | 3 |
| John Doe | 3 |

--7) Which product has been bought the least in terms of quantity?

SELECT product\_name AS product, MIN(quantity) AS quantity

FROM order\_items oi

JOIN products p

ON oi.product\_id = p.product\_id

WHERE quantity <= 1

GROUP BY product\_name

ORDER BY quantity

Explanation:

The query selects the **product\_name** and uses the **MIN()** function to calculate the minimum quantity for each product.

The results are grouped by **product\_name** and ordered by quantity in ascending order.

Output:

|  |  |
| --- | --- |
| product | quantity |
| Product A | 1 |
| Product B | 1 |
| Product C | 1 |
| Product D | 1 |
| Product E | 1 |
| Product G | 1 |
| Product H | 1 |
| Product I | 1 |
| Product K | 1 |
| Product L | 1 |

--8) What is the median order total?

WITH CTE\_order AS (

SELECT SUM(p.price \* oi.quantity) AS order\_total,

ROW\_NUMBER() OVER (ORDER BY SUM(p.price \* oi.quantity)) AS row\_num,

COUNT(\*) OVER () AS total\_orders

FROM order\_items oi

JOIN products p

ON oi.product\_id = p.product\_id

GROUP BY oi.order\_id

)

SELECT order\_total AS median\_order\_total

FROM CTE\_order

WHERE row\_num = (total\_orders + 1) / 2

Explanation:

The query calculates the order total for each order by multiplying the price and quantity for each order item using the **SUM()** function.

It uses the **ROW\_NUMBER()** function to assign a row number to each order total when sorted in ascending order.

The result selects the order total corresponding to the row number equal to **(total\_orders + 1) / 2**, representing the middle value or median.

Output:

|  |
| --- |
| median\_order\_total |
| 85 |

--9) For each order, determine if it was ‘Expensive’ (total over 300), ‘Affordable’ (total over 100), or ‘Cheap’.

SELECT oi.order\_id, SUM(p.price \* oi.quantity) AS order\_total,

CASE

WHEN SUM(p.price \* oi.quantity) > 300 THEN 'Expensive'

WHEN SUM(p.price \* oi.quantity) > 100 THEN 'Affordable'

ELSE 'Cheap'

END AS order\_category

FROM order\_items oi

JOIN products p

ON oi.product\_id = p.product\_id

GROUP BY oi.order\_id

Explanation:

The query calculates the order total by multiplying the price and quantity for each order item using the **SUM()** function.

It uses the **CASE** statement to assign a category ('Expensive', 'Affordable', or 'Cheap') based on the order total.

Output:

|  |  |  |
| --- | --- | --- |
| order\_id | order\_total | order\_category |
| 1 | 35 | Cheap |
| 2 | 75 | Cheap |
| 3 | 50 | Cheap |
| 4 | 80 | Cheap |
| 5 | 50 | Cheap |
| 6 | 55 | Cheap |
| 7 | 85 | Cheap |
| 8 | 145 | Affordable |
| 9 | 140 | Affordable |
| 10 | 285 | Affordable |
| 11 | 275 | Affordable |
| 12 | 80 | Cheap |
| 13 | 185 | Affordable |
| 14 | 145 | Affordable |
| 15 | 225 | Affordable |
| 16 | 340 | Expensive |

--10) Find customers who have ordered the product with the highest price.

WITH CTE\_max\_price AS (

SELECT MAX(price) AS max\_price

FROM products

)

SELECT CONCAT(c.first\_name, ' ', c.last\_name) AS customer, p.product\_name AS product, p.price

FROM customers c

JOIN orders o

ON c.customer\_id = o.customer\_id

JOIN order\_items oi

ON o.order\_id = oi.order\_id

JOIN products p

ON oi.product\_id = p.product\_id

CROSS JOIN CTE\_max\_price mp

WHERE p.price = mp.max\_price

Explanation:

The query uses a CTE to calculate the maximum price from the **products** table.

It joins the **customers**, **orders**, **order\_items**, and **products** tables to get the customer details for orders that match the product with the highest price.

The result includes the customer name, product name, and price for the product with the highest price.

Output:

|  |  |  |
| --- | --- | --- |
| customer | product | price |
| Ivy Jones | Product M | 70 |
| Sophia Thomas | Product M | 70 |